M13-L2 Problem 1

Once more, we will study the stress prediction problem, this time using XGBoost, a very powerful boosting method.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error


def plot_shape(dataset, index, model=None, lims=None):
    x = dataset["coordinates"][index][:,0]
    y = dataset["coordinates"][index][:,1]

    if model is None:
        c = dataset["stress"][index]
    else:
        c = model.predict(dataset["features"][index])

    if lims is None:
        lims = [min(c),max(c)]

    plt.scatter(x,y,s=5,c=c,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75, pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_shape_comparison(dataset, index, model, title=""):
    plt.figure(figsize=[6,3.2], dpi=120)
    plt.subplot(1,2,1)
    plot_shape(dataset,index)
    plt.title("Ground Truth",fontsize=9,y=.96)
    plt.subplot(1,2,2)
    c = dataset["stress"][index]
    plot_shape(dataset, index, model, lims = [min(c), max(c)])
    plt.title("Prediction",fontsize=9,y=.96)
    plt.suptitle(title)
    plt.show()

def load_dataset(path):
    dataset = np.load(path)
    coordinates = []
    features = []
    stress = []
    N = np.max(dataset[:,0].astype(int)) + 1
```

```
    split = int(N*.8)
    for i in range(N):
        idx = dataset[:,0].astype(int) == i
        data = dataset[idx,:]
        coordinates.append(data[:,1:3])
        features.append(data[:,3:-1])
        stress.append(data[:,-1])
    dataset_train = dict(coordinates=coordinates[:split], features=features[:split], stress=stress[:split])
    dataset_test = dict(coordinates=coordinates[split:], features=features[split:], stress=stress[split:])
    X_train, X_test = np.concatenate(features[:split], axis=0), np.concatenate(features[split:], axis=0)
    y_train, y_test = np.concatenate(stress[:split], axis=0), np.concatenate(stress[split:], axis=0)
    return dataset_train, dataset_test, X_train, X_test, y_train, y_test

def get_shape(dataset,index):
    X = dataset["features"][index]
    y = dataset["stress"][index]
    return X, y

def eval_model(model, verbose=False):
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)
    mse_train = mean_squared_error(y_train, pred_train)
    mse_test = mean_squared_error(y_test, pred_test)
    if verbose:
        print(f"Train MSE = {mse_train:.2e}")
        print(f"Test  MSE = {mse_test:.2e}")
    return mse_train, mse_test
```

Loading the data

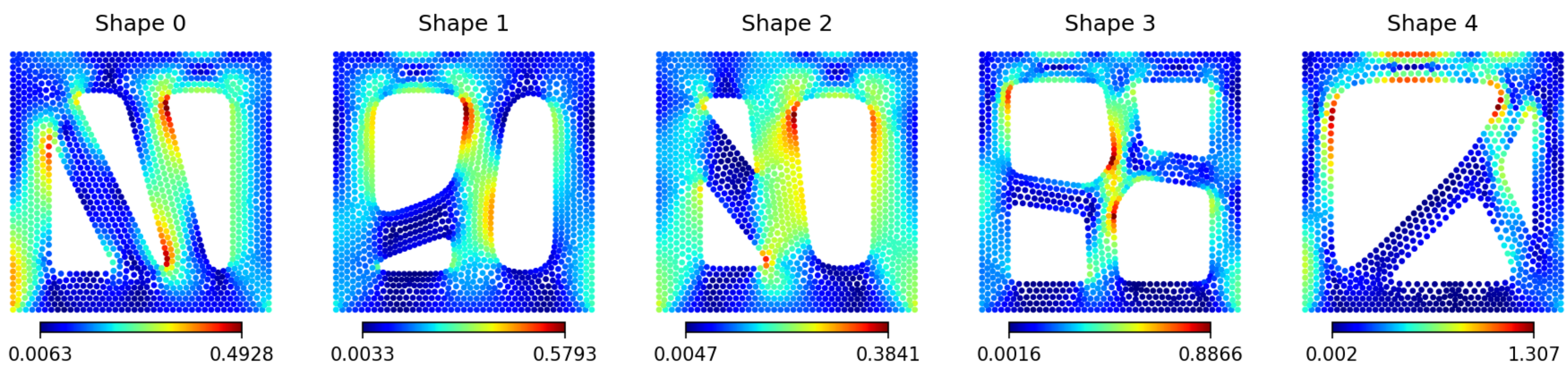
First, complete the code below to load the data and plot the von Mises stress fields for a few shapes.
You'll need to input the path of the data file, the rest is done for you.

All training node features and outputs are in `X_train` and `y_train` , respectively. Testing nodes are in `X_test` , `y_test` .

`dataset_train` and `dataset_test` contain more detailed information such as node coordinates, and they are separated by shape. Get features and outputs for a shape by calling `get_shape(dataset,index)` . `N_train` and `N_test` are the number of training and testing shapes in each of these datasets.

In [3]:
```
# YOU MAY NEED TO EDIT data_path
data_path = "data/stress_nodal_features.npy"
dataset_train, dataset_test, X_train, X_test, y_train, y_test = load_dataset(data_path)
N_train = len(dataset_train["stress"])
N_test = len(dataset_test["stress"])

plt.figure(figsize=[15,3.2], dpi=150)
for i in range(5):
    plt.subplot(1,5,i+1)
    plot_shape(dataset_train,i)
    plt.title(f"Shape {i}")
plt.show()
```



XGBoost Regressor

XGBoost models, like `XGBRegressor` here, can be used much like sklearn models.

First, define an instance of `XGBRegressor` with the desired parameters; then, fit the model with `model.fit` . You can evaluate a fitted model with `model.predict` .

The provided function `mse_train, mse_test = eval_model(model)` to get MSE values on the train and test datasets.

In [4]:
```
eta = 0.8
depth = 9

params = dict(
    eta = eta,
    max_depth = depth,
)

model = XGBRegressor(objective ='reg:squarederror', seed = 123, n_estimators = 10, **params)
model.fit(X_train, y_train)

mse_train, mse_test = eval_model(model)
print("  eta   depth   |   Train MSE    Test MSE")
print("----------------|-------------------------")
print(f"  {eta:.1f}    {depth:>2d}     |    {mse_train:.2e}    {mse_test:.2e}")
```

```
  eta   depth   |   Train MSE    Test MSE
----------------|-------------------------
  0.8     9     |    2.02e-03    6.22e-03
```

Parametric study

Now let's examine the effects of varying the parameters `eta` and `max_depth` , keeping `n_estimators` as 10. For every combination of eta in [0.1, 0.3, 0.5, 0.7] and `max_depth` in [5, 10, 15, 20], train an XGB regressor and report the train and test MSE values.

Which combination has the best performance on testing data?

In [5]:
```
# YOUR CODE GOES HERE
etas = [0.1,0.3,0.5,0.7]
max_depths = [5,10,15,20]

for eta in etas:
    for depth in max_depths:
        model = XGBRegressor(objective ='reg:squarederror', seed = 123, n_estimators = 10, eta = eta, max_dep
        model.fit(X_train, y_train)
        mse_train, mse_test = eval_model(model)
        print("  eta   depth   |   Train MSE    Test MSE")
        print("----------------|--------------------------")
        print(f"  {eta:.1f}    {depth:>2d}     |    {mse_train:.2e}    {mse_test:.2e}")
        print("\n")
```

```
 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.1     5      |    2.28e-02    2.47e-02


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.1    10      |    1.81e-02    2.07e-02


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.1    15      |    1.64e-02    1.98e-02


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.1    20      |    1.60e-02    2.00e-02


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.3     5      |    5.47e-03    7.27e-03


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.3    10      |    1.65e-03    4.77e-03


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.3    15      |    4.07e-04    4.65e-03


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.3    20      |    2.29e-04    4.84e-03


 eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
 0.5     5      |    4.63e-03    6.56e-03
```

```
  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.5    10      |    1.38e-03    4.75e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.5    15      |    2.02e-04    5.13e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.5    20      |    2.40e-05    5.39e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.7     5      |    5.02e-03    6.88e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.7    10      |    1.53e-03    5.78e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.7    15      |    2.56e-04    6.07e-03


  eta   depth    |   Train MSE    Test MSE
----------------|-------------------------
  0.7    20      |    3.31e-05    6.58e-03
```

Eta = 0.3 and depth = 10 gives the best results i.e., least MSE on the testing data.